Discovering Deadlocks in a Memory Controller IP

Jef Verdonck, Emrah Armagan, Khaled Nsaibia, Slava Bulach – u-blox AG

Pranay Gupta, Anshul Jain, Chirag Agarwal, Roger Sabbagh – Oski Technology

accellera
SYSTEMS INITIATIVE

2020
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE

Agenda

- u-blox deadlock verification challenge
- Formal methodology for discovering deadlocks
- Memory Controller case study
- Results
- Conclusions

About u-blox

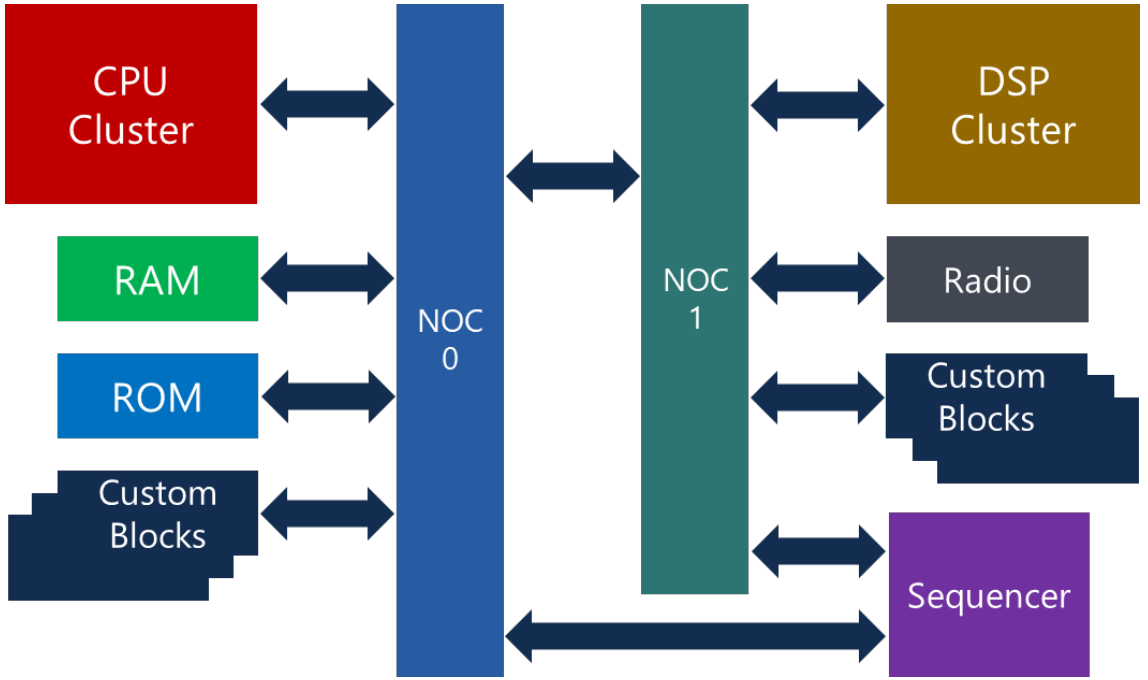
- Global provider of leading positioning and wireless communication technologies
- u-blox enables OEMs to reliably locate and connect people and devices
- A fabless company owning the full IP focusing on R&D and customer relationships
- All manufacturing outsourced
- Founded in 1997 as a spin-off from Swiss Federal Institute of Technology

Products

	P Positioning	C Cellular Communication	S Short Range Communication
Integrated Circuits			
Modules			
U Services	Connectivity Core Protection Extensions		

Design Deadlocks

- Deadlock sources
  - Cyclical dependency
  - Mutually blocking processes
  - Forever waiting for resources
- SoC-level and IP-level deadlocks
- SW application dependent

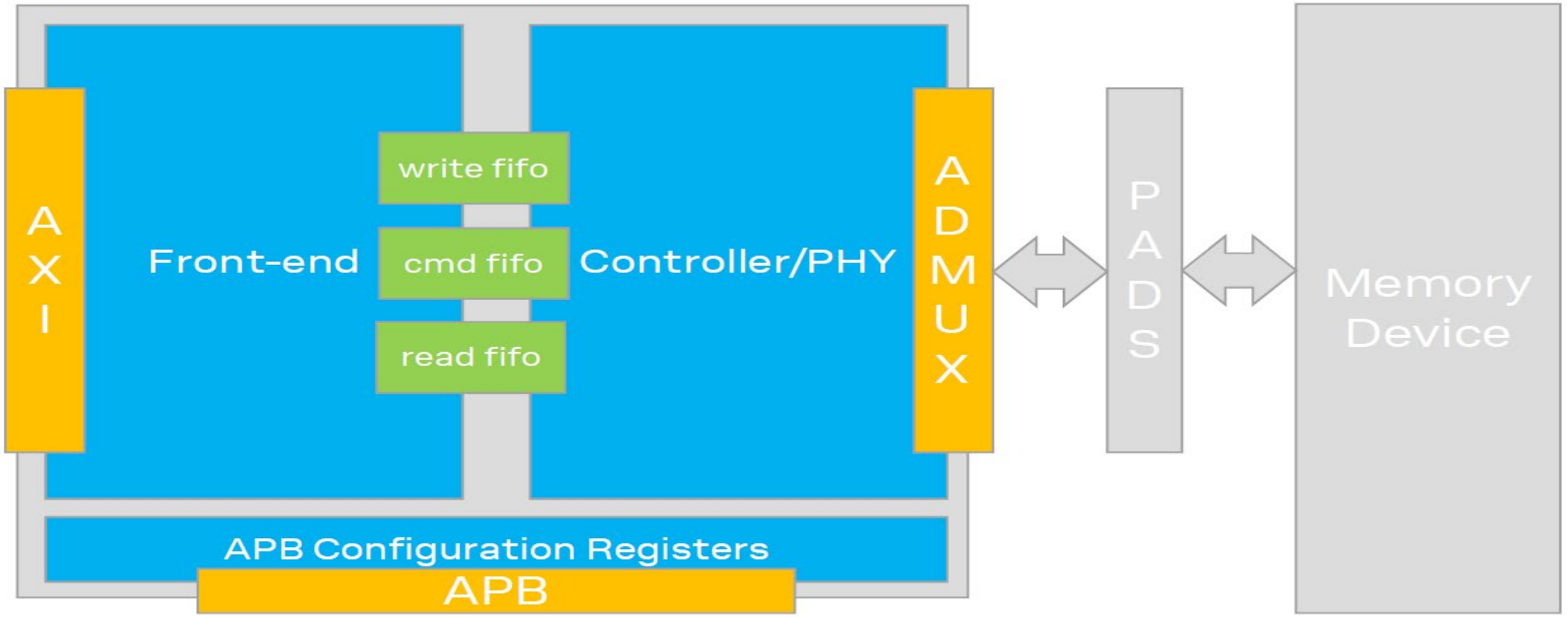


Dynamic Deadlock Verification Challenge

- Deadlocks are not directly targeted
  - Simulation
  - Emulation
  - FPGA prototyping
- Rare scenarios
- Coverage not a guarantee

Image sources:
1. https://upload.wikimedia.org/wikipedia/en/thumb/e/ef/SystemVerilog_logo.png/240px-SystemVerilog_logo.png
2. https://media.digikey.com/Photos/Xilinx%20Photos/MFG_CK-V7-VC7215-G-J.jpg
3. https://www.accellera.org/images/community/uvm/uvm_logo.png
4. https://www.cadence.com/content/dam/cadence-www/global/en_US/diagrams/tools/system-design-verification/palladium-z1.jpg

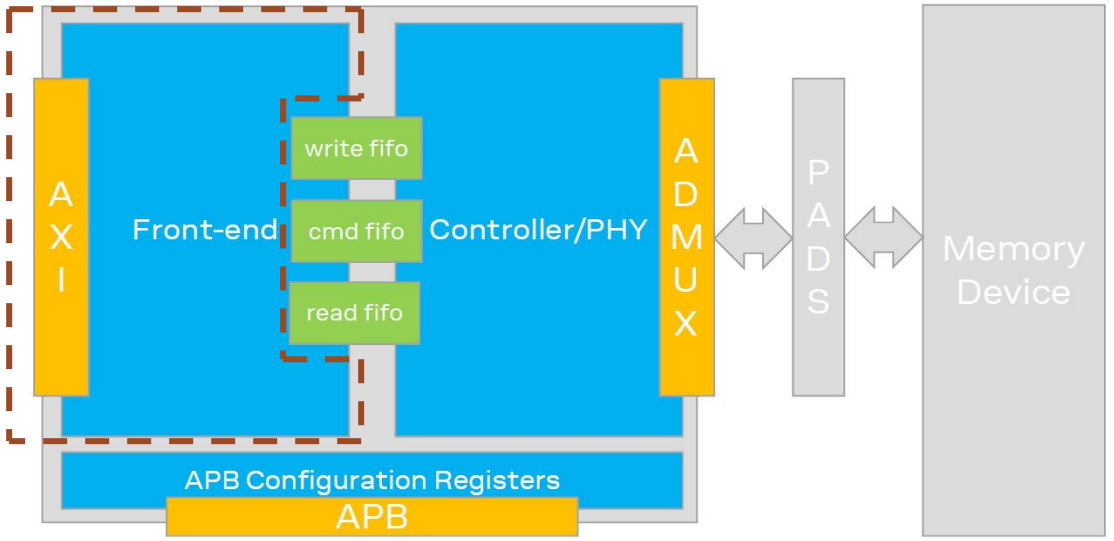
Memory Controller IP



Memory Controller Sources of Deadlock

Three primary components:

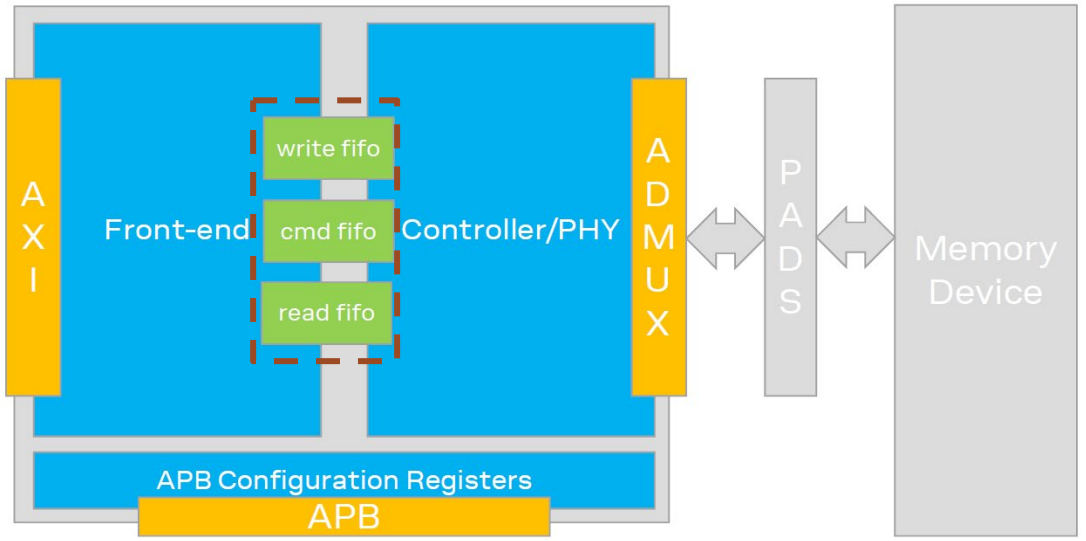
1. AXI Front-End (AXI-FE)
   - Read path stalls due to RAW logic
   - Spurious Read requests
   - Incorrect volume of Read data requested
2. Asynchronous Queues
3. Controller/PHY



Memory Controller Sources of Deadlock

Three primary components:

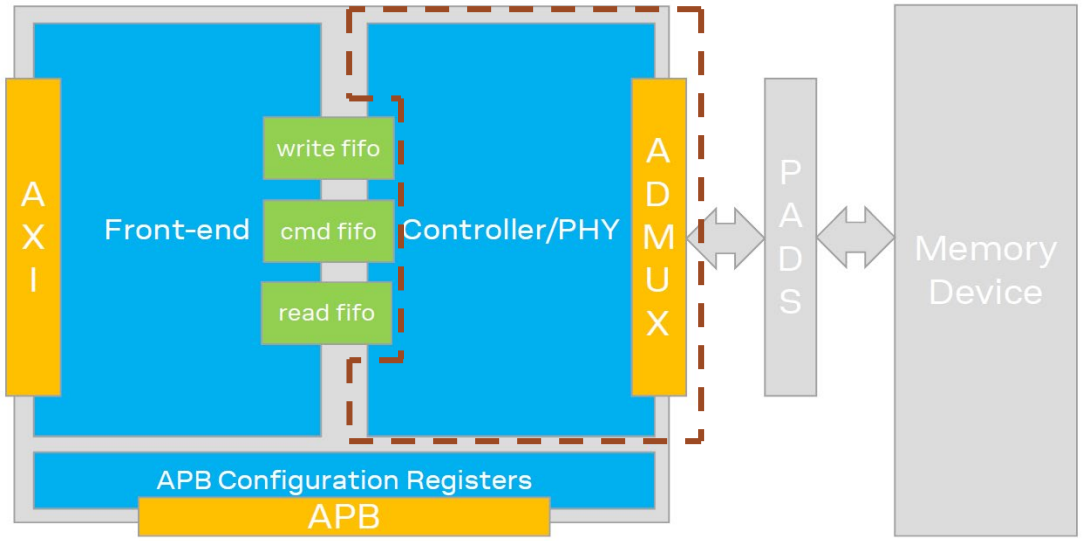
1. AXI Front-End (AXI-FE)
2. Asynchronous Queues
   - Commands corrupted
   - Data not cleared out
   - Data dropped
3. Controller/PHY



Memory Controller Sources of Deadlock

Three primary components:

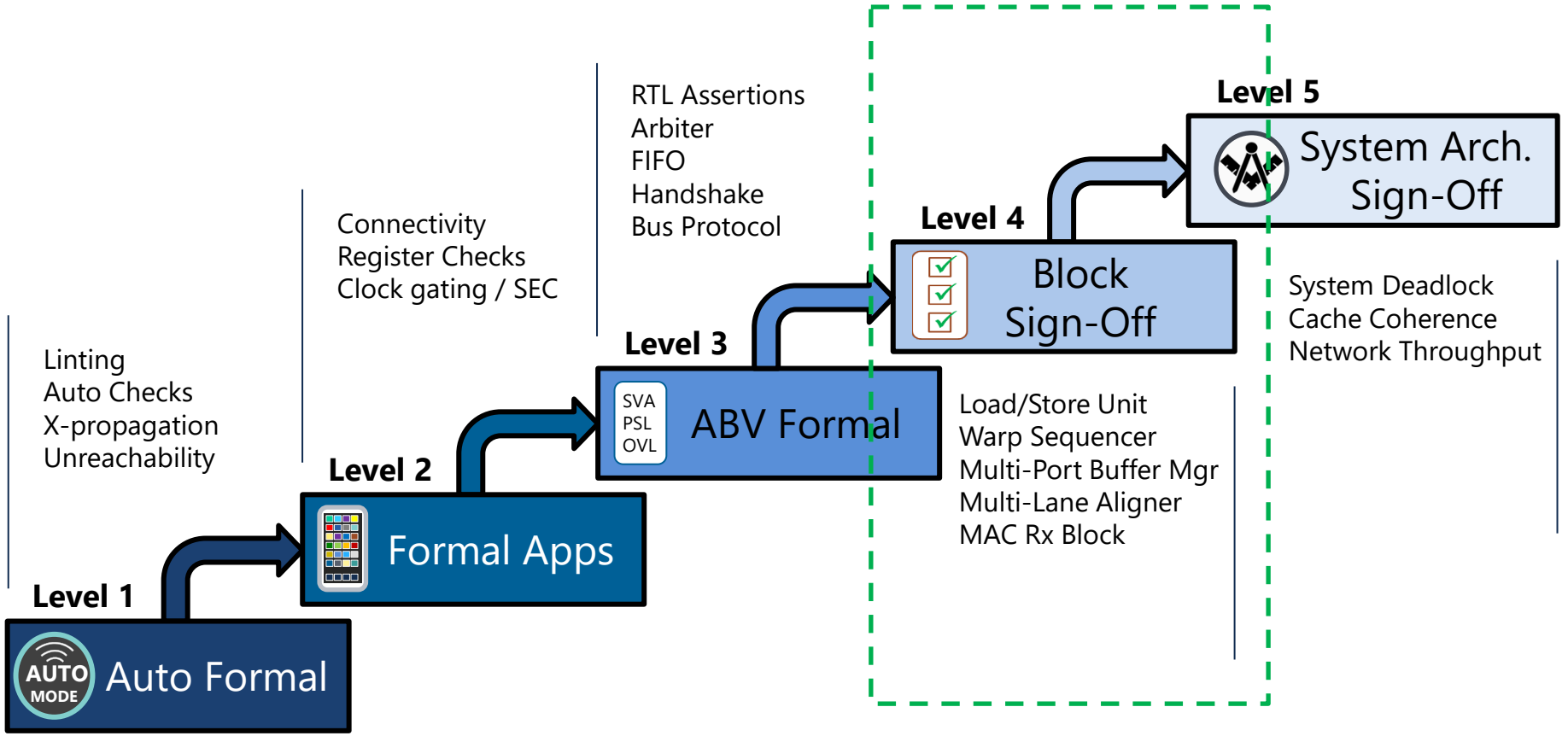
1. AXI Front-End (AXI-FE)
2. Asynchronous Queues
3. Controller/PHY
   - Commands dropped
   - Extra Read data returned
   - Read data dropped



Agenda

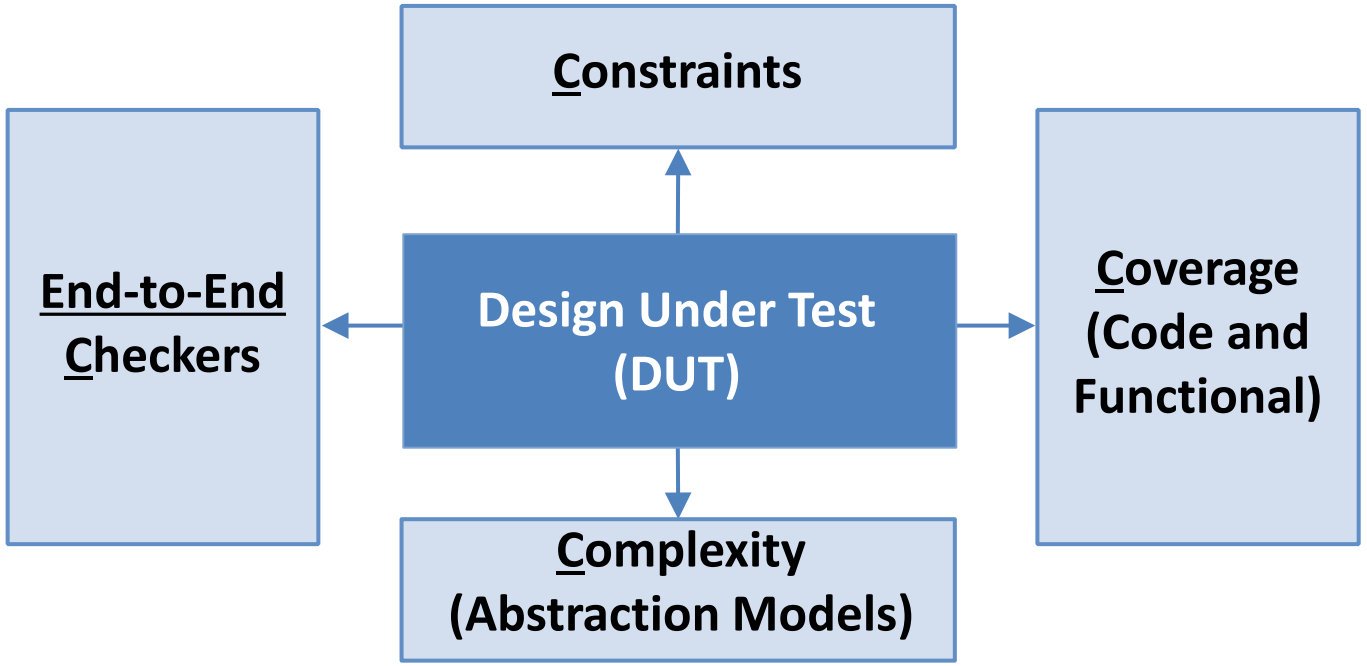
- u-blox deadlock verification challenge
- Formal methodology for discovering deadlocks
- Memory Controller case study
- Results
- Conclusions

Formal Capability Maturity Model



Level 4 Formal Methodology

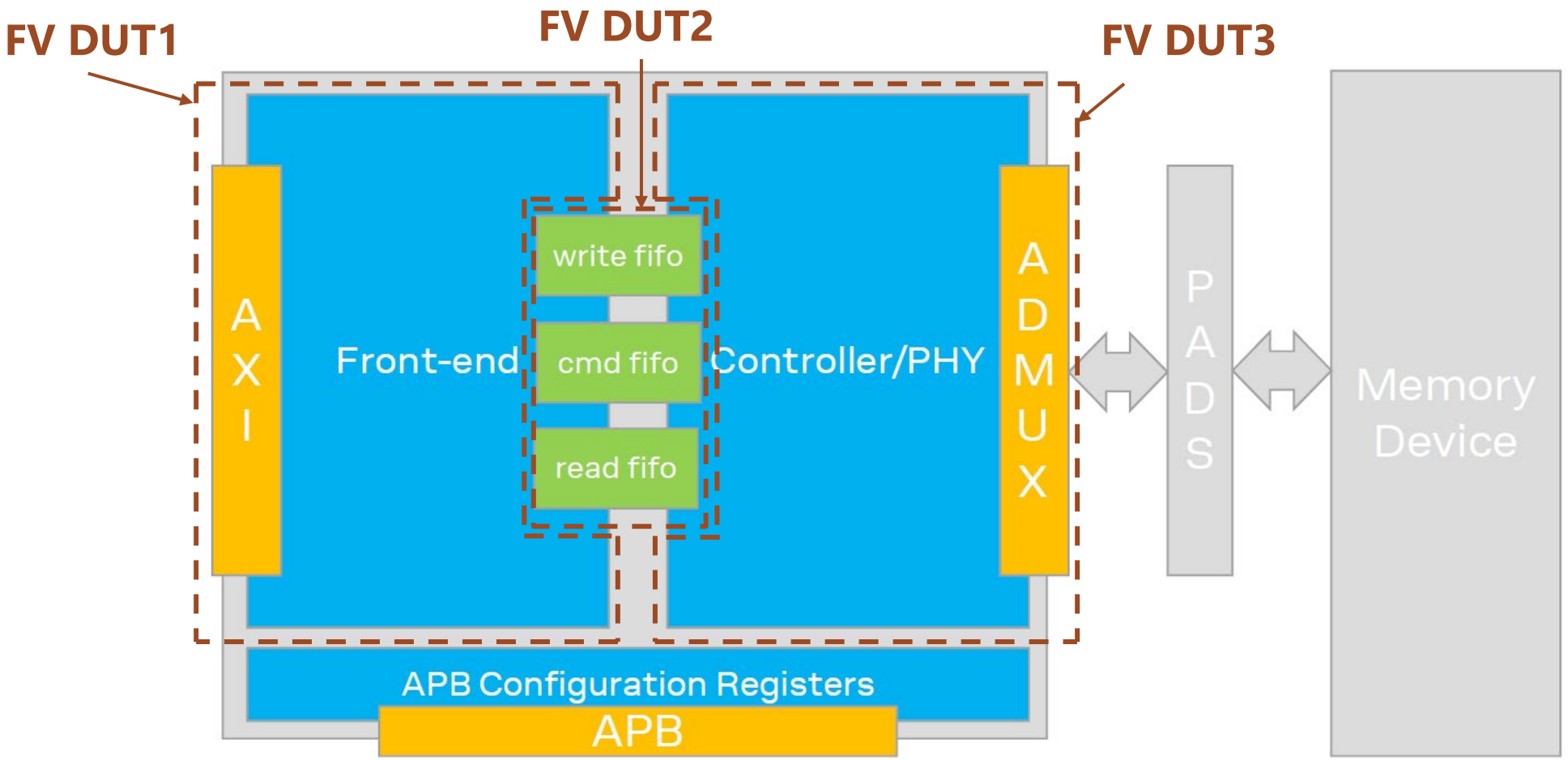
Quality of results depends on all 4 Cs



Agenda

- u-blox deadlock verification challenge
- Formal methodology for discovering deadlocks
- Memory Controller case study
- Results
- Conclusions

Partitioning for Formal Verification



AXI FE Deadlock Checkers

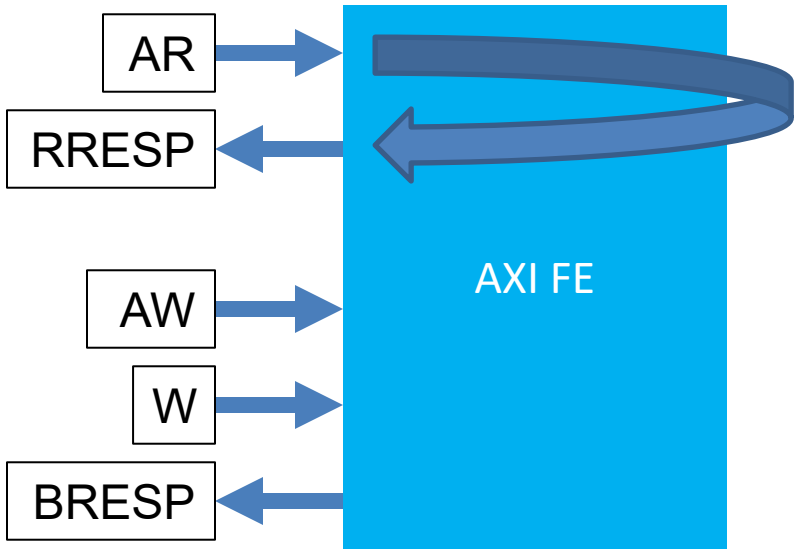
- Forward progress checkers for Reads
  - If there are any pending read requests, then RVALID should assert within a finite time duration
  - If there are RAW hazard conditions, the RAW hazard should resolve within a finite time duration
  - There should be a finite number of read requests in the pipeline of the design at any given time
- Command correctness checkers
  - There should be no spurious command sent to the Async CMD queue for an address which does not match to a transaction on the AXI interface
  - For a Read request, the number of words requested by the command sent to the Async CMD queue should be the same as the request received at the AXI interface

Forward progress checker

- If there are any pending read requests, then RVALID should assert within a finite time duration
  - Finite duration can be called cycle timeout which will depend on the design pipelines
  - In this application, finite duration will depend on max number of cycle to send read data for accepted read request

- Logic for calculating time (restart and stalling conditions)

```
always @ (posedge clk) begin
  if (rst) cyc_cnt <= 'd0;
  else if (pending_rd_req == 'd0) cyc_cnt <= 'd0;
  else if (rvalid) cyc_cnt <= 'd0;
  else if (back_pressure_from_mem) cyc_cnt <= cyc_cnt;
  else if (raw_hazard & wr_req_waiting_for_wdata) cyc_cnt <= cyc_cnt;
  else cyc_cnt <= cyc_cnt + 'd1;
end
```



- Property Implemented: where TIMEOUT is parameter

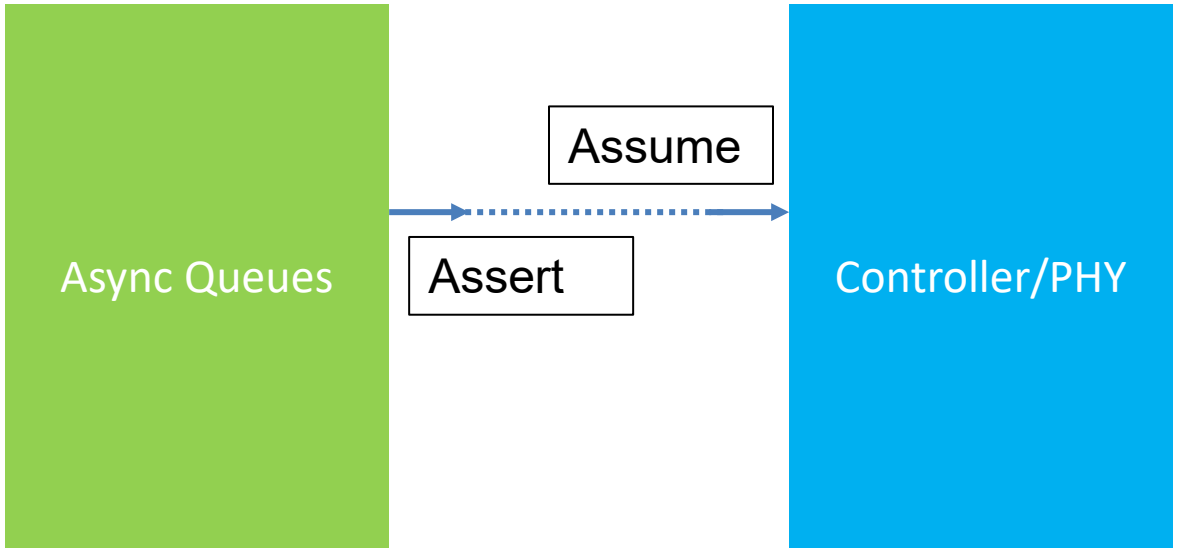
```
ar2rresp_read_fwd_progress:
assert property (
  @ (posedge clk) disable iff (rst)
    (cyc_cnt == TIMEOUT) |-> (rvalid)
);
```

Async Queues Checkers

- Command Queue
  - Forward progress – commands pushed in should be popped out within a finite time duration
  - Data transport – there should be a one-to-one relationship between commands pushed in and commands popped out
- Write Data Queue
  - Forward progress – data pushed in should be popped out within a finite time duration
  - Data transport – there should be a one-to-one relationship between data pushed in and data popped out
  - Pop interface protocol – when a pop is requested from the queue, all of the requested data for the corresponding command should be available at the pop side of the queue
- Read Data Queue
  - Data transport – there should be a one-to-one relationship between data pushed in and data popped out

Constraints Validation

- Inputs to each DUT are constrained with assumptions
- Assumptions used as asserts on outputs of adjoining blocks
- Run in simulation or formal
- A failure indicates:
  - Over-constraint on the DUT
  - Bug in the adjoining block

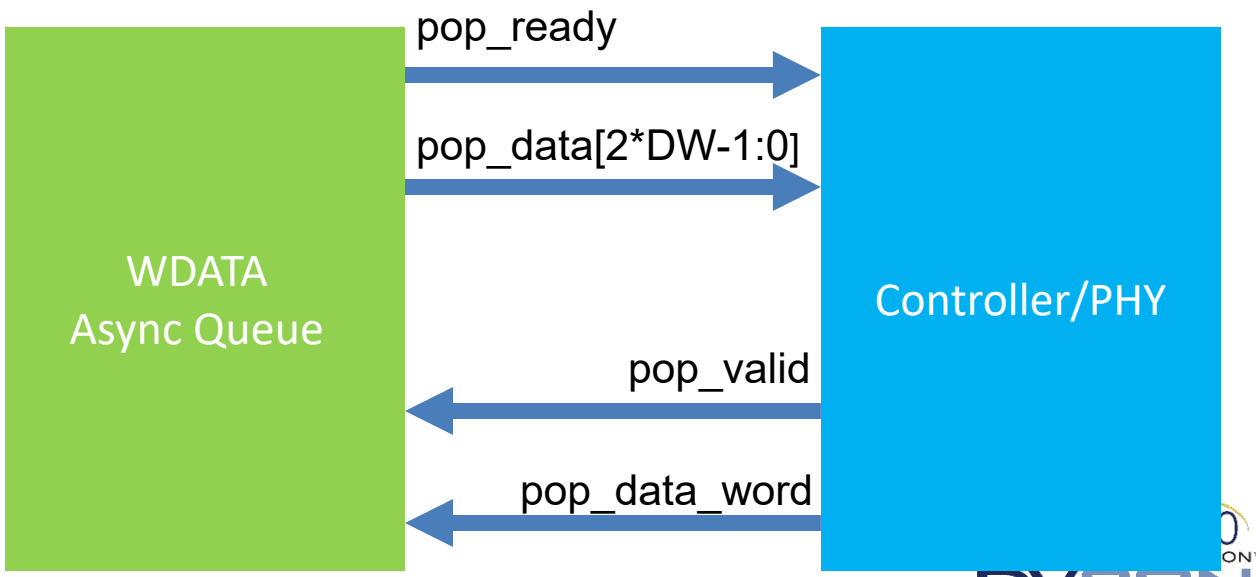


Constraints Validation – Pop Interface Protocol

- When the pop is requested by Controller/PHY, write data words for the write command should be available at the pop side of WDATA Async Queue
- `pop_ready = occupancy_wdata_asyncq > pop_data_word`

- Property Implemented:

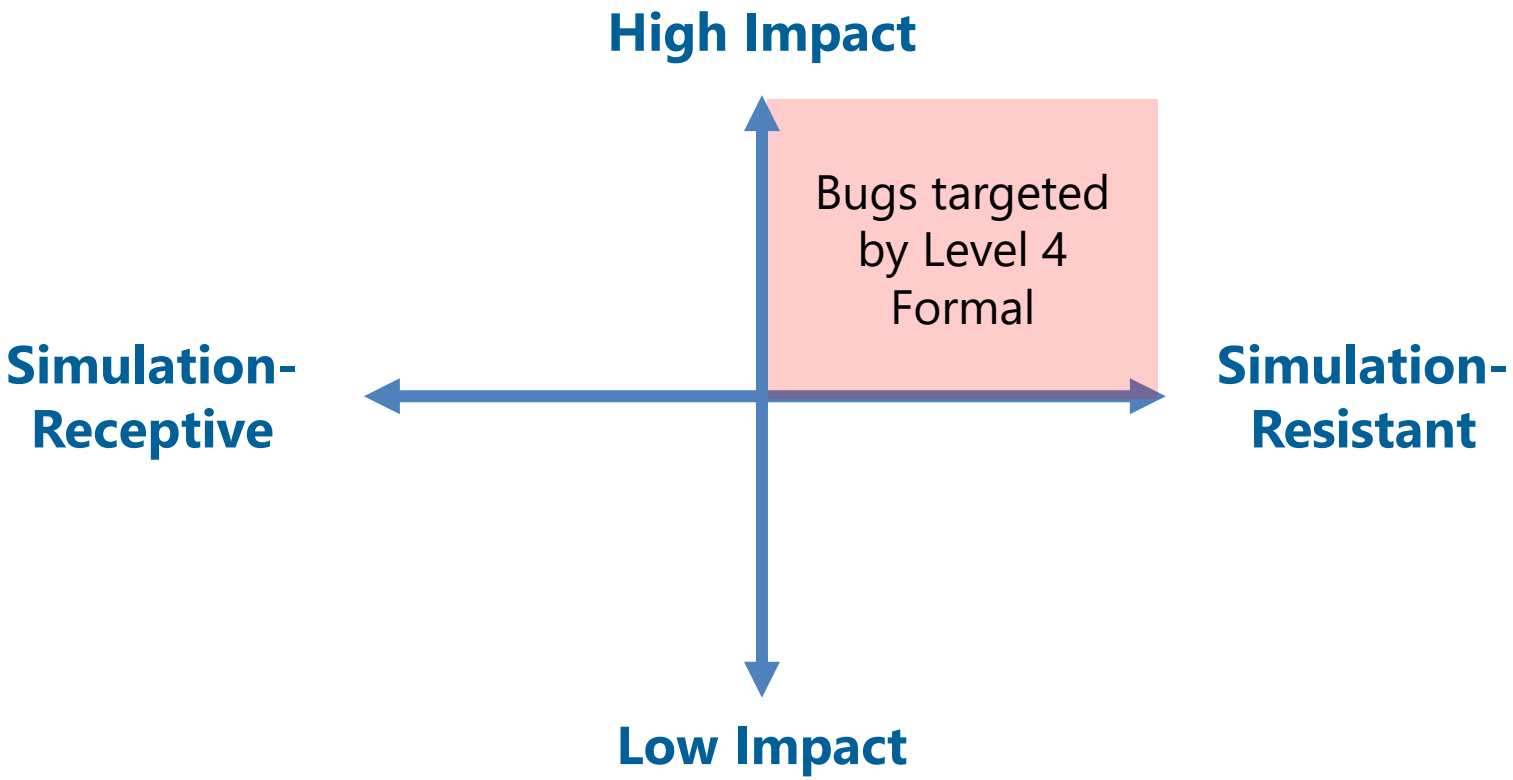
```
wdataq2phy_pop_rdy_when_pop_valid:
assert property (
  @ (posedge clk) disable iff (rst)
    (pop_valid) |-> (pop_ready)
);
```



Agenda

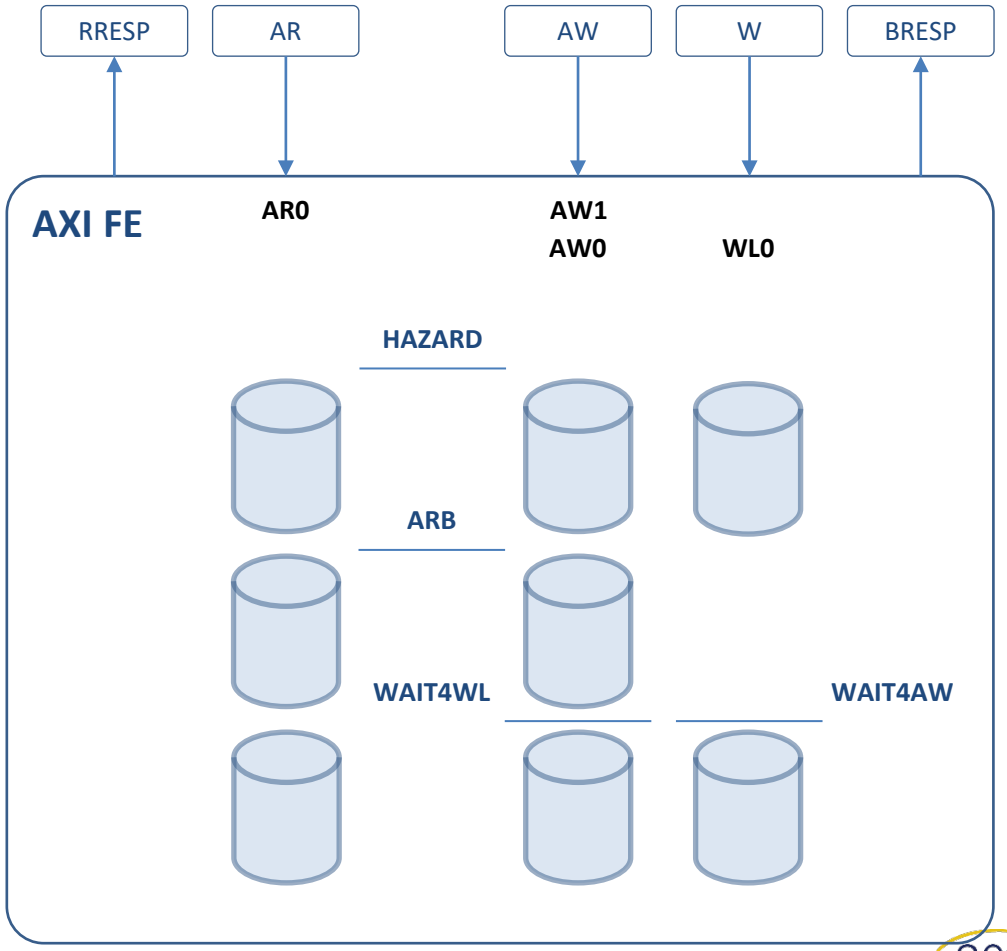
- u-blox deadlock verification challenge
- Formal methodology for discovering deadlocks
- Memory Controller case study
- Results
- Conclusions

Classification of Bugs

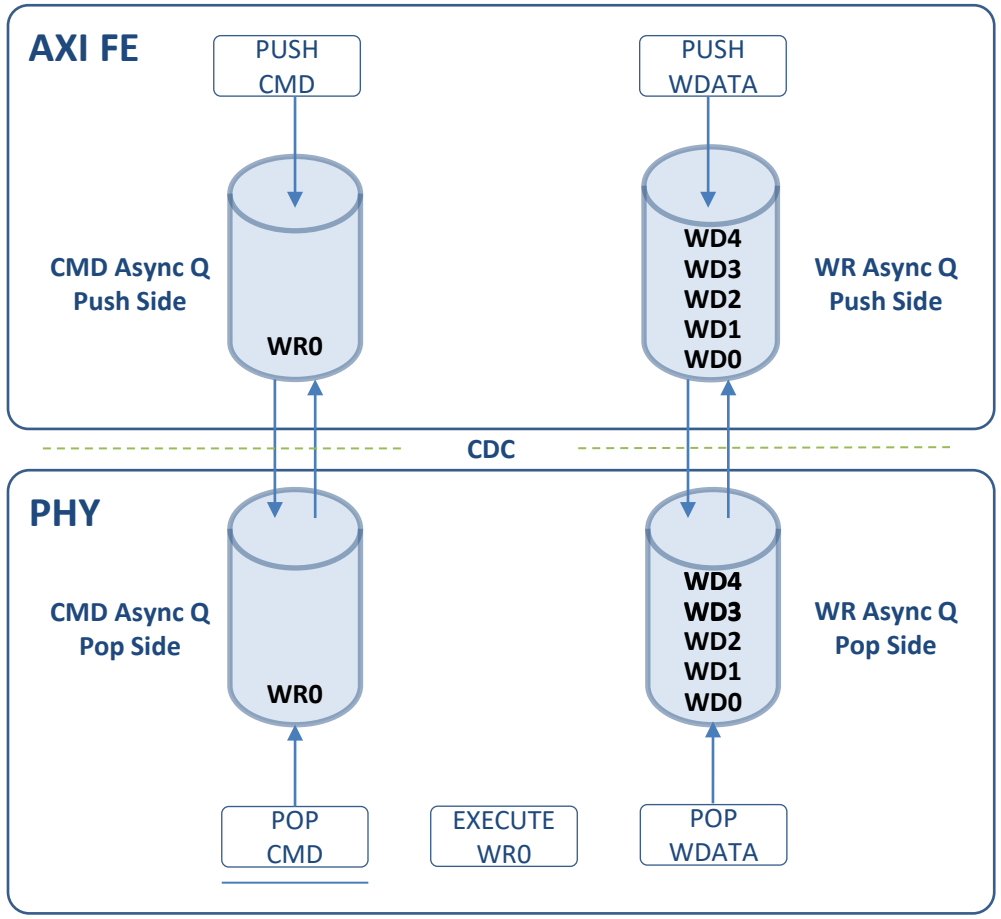


Read-after-Write Hazard Deadlock

1. AW channel received a write request (AW0)
2. AW0 request is blocked until all write data for AW0 is received
3. AR channel receives an overlapping read request (AR0)
4. RAW Hazard is detected between AW0 and AR0
   - AW0 is still waiting for the remaining write data on the W channel and AR0 is waiting for the hazard to resolve (i.e. AW0 write request to move forward)
5. AW channel receives *another* overlapping write request (AW1)
6. W channel channel receives all data for the AW0 request
   - Therefore, validating AW0 to move forward in the design
7. After the AW0 request moves forward, the RAW hazard is again detected
   - This time between AW1 and AR0, even though the AR0 request is an **older** read



Orphaned Write Data in Async Queue Deadlock



1. PHY pops a Write Command (WR0) with length of 5 data words
   - When all write data is not reflected at pop side, but all write data for the command is available at push side
2. PHY sends a pop request to Write Data Queue to fetch 2 out of the 5 data words
   - However, the occupancy of the Write Data Queue on the pop side is currently ***only 1 data*** word which violates the interface protocol
3. PHY samples 1 word of real data and 1 word of garbage data
4. Write Data Queue pop side pointer is then updated with the balance of the 4 words in the burst
5. PHY then reads the remaining 3 words, which leaves 1 word orphaned in the Write Data Queue after completion of the command

Agenda

- u-blox deadlock verification challenge
- Formal methodology for discovering deadlocks
- Memory Controller case study
- Results
- Conclusions

Conclusions

- Level 4 formal methodology
  - Effective at finding simulation-resistant deadlock scenarios
- Efficient use of resources to find hidden deadlocks
  - 1 month of FV engineering effort finds bugs undetected after many months and millions of cycles of dynamic testing

Thank you!
Questions?